Towards Automated Context-aware Software Quality Management

Gregor Grambow and Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
{gregor.grambow, roy.oberhauser}@htw-aalen.de

***Abstract* — To consistently improve software quality management, greater automation and tighter integration of quality tools and measurements in the software engineering environment is essential. However, automation of software quality management faces numerous challenges such as project uniqueness, project dynamics, efficiency, and limited time and quality expenditures. In this paper, an approach is proposed that extends the Goal-Question-Metric technique and automates the monitoring of quality goals via a multi-agent system by using competitive bidding agent behavior for proactive vs. cooperative voting for reactive measures. The preliminary results show promise for systematically harmonizing (conflicting) quality attributes, goals, metrics, and countermeasures and for automating aspects of software quality management.**

Keywords-software quality management; agents; Goal-Question-Metric technique; automated software engineering; software engineering environments

I. INTRODUCTION

Software quality management (SQM), which addresses qualities in the software product as well as its development processes, faces ongoing challenges. Essential difficulties inherent in software such as its invisibility, complexity, conformity, and changeability [4] make SQM more difficult than quality management in other disciplines. Progress is not objectively measureable, development is not deterministic, and frequent change and refinement occurs in processes, best practices, quality techniques, programming languages, development tools, available metrics, team collaboration, etc.

On the one hand, development processes are defined at a relatively high generic level, and software engineers are faced with a large set of expectations, for instance in order to meet specifications for the development process (e.g., CMMI (Capability Maturity Model Integration), ISO 15504, ISO 9001) and for the product (e.g., standards, requirements) within given project constraints. On the other hand, every software engineer is faced with low-level micro decisions about which activities or task sequences are performed when.

While both preventative and analytical quality measures are part of the standard SQM repertoire, to be effective their relevance, degree of usage, and temporal suitability must be considered. While preventative measures can reduce the number and cost of future failures, some combination with reactive measures is necessary for achieving quality. Determining the appropriate time for the appropriate person(s) to apply the appropriate technique to the appropriate artifact is a challenge, and cannot readily be determined and pre-planned far in advance due to lack of information and changing circumstances.

Additionally, these quality measures need to remain aligned to the various project-specific product-and-process quality goals throughout the development lifecycle. Considering just the quality attributes in software products, [19] studied 24 ATAM (Architecture Tradeoff Analysis Method) evaluations and found a significantly varied distribution of quality attributes. When one considers various aspects influencing quality attributes such as project-specific quality distributions, stakeholder differences and preferences, developer (subconscious and temporal) prioritization (based on personality, experience, culture, stakeholder subset contact, miscommunication, etc.), forgetfulness, incomplete checklists, etc., then it is not surprising if a deviation exists between the actual and intended/expected effects of quality goals and measures.

Effective SQM decision-making requires effective and cost-efficient metrics that are retrieved, utilized, and aligned to goals. One well-known technique for this is Goal-Question-Metric (GQM) [1]. The model consists of a hierarchical structure that starts with a goal, refines it into several questions, each of which is then refined into metrics. In turn, these metrics are used for data collection, analysis of the answers to the questions, and then determination of goal achievement. The technique is typically applied manually and can be applied to the process as well as product level.

Based on GQM results and goal divergence, appropriate measures should be applied in a timely fashion in accordance with the analysis. Due to the dynamic nature of software projects, this process should be continuous and therefore automated in order to be effective and efficient.

In the face of these challenges, this paper presents an automated approach that extends GQM and continually monitors quality goals via a multi-agent system (MAS). The system provides measure selection guidance for code quality measures to developers on-the-fly in their software engineering environment (SEE). Thus, the quality of the product is not only continuously measured (e.g., via static analysis), but also constantly addressed via measures relating to detected problems.

The remainder of this paper is organized as follows: Section II describes the solution approach, Section III concretizes the concept and goes into implementation details, followed in Section IV by an evaluation. Section V describes related work, which is then followed by a conclusion.

II. SOLUTION APPROACH

The Context-aware Software Engineering Environment Event-driven framework (CoSEEEK) [17], which provides the infrastructure for the solution, will now be discussed. The conceptual architecture is shown in Figure 1.

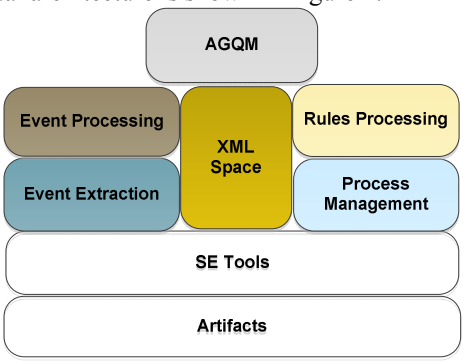

Figure 1. CoSEEEK Conceptual Architecture

SE Tools is a placeholder for heterogeneous development and testing tools. *Event Extraction* consists primarily of event sensors and data collection for SE tools. *XML Space* provides event and data storage in a loosely coupled fashion via an XML-based tuple space implementation. *Event Processing* applies complex event processing (CEP) and any contextual annotation to events. *Process Management* is a workflow engine aware of and responsible for SE process conformance of activities and supports adaptive task management and guidance to developers via their IDE (Integrated Development Environment). *Rules Processing* consists of a rule engine that analyzes static analysis reports, metrics, etc. and triggers events as necessary. The *AGQM* (Automated GQM) module provides a MAS with behavior agents that support an extended and automated GQM.

A. GQM Extensions

Two main requirements have to be satisfied to facilitate automatic support for GQM execution. Firstly, a GQM plan must exist that defines the relations between goals, questions, and metrics. Secondly, the metrics have to be integrated in the SEE, enabling the automatic extraction of metrics and thus the automatic receipt of possible deviation information.

Some extensions to the GQM technique were necessary to support automation. Different abstractions of key performance indicators (KPIs) were introduced to enable automatic calculation of goal deviations. Metrics are encapsulated in KPIs to enable consolidation and simplified deviation calculation. Since multiple metrics may be utilized for a single question in GQM, a QKPI (Question KPI) was created for consolidation at the question level. Similarly for goals, multiple questions may apply to a single goal, thus a GKPI (Goal KPI) is used for goal deviation calculations. Formulas for each of the KPIs specify how metrics are combined. To support automated multiple goal attainment, each defined goal was assigned one agent responsible for monitoring and fulfillment of that goal.

Figure 2 shows the relation between the different conceptual elements. In the implementation, the GQM plan containing these relations is defined in XML.

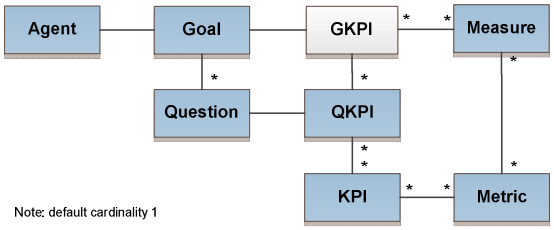

Figure 2. AGQM data structure

To prescribe appropriate countermeasures for (potential) quality deterioration, measures were categorized as follows: reactive (or analytical) measures, which are directly associated with concrete metrics or violations; and proactive (or preventative) measures, which hereby are categorized as supporting certain quality goals at an abstract level and may not be readily associated with a concrete problem. Proactive measures are assigned to GKPIs and can be triggered either when a GKPI deviation occurs (a supportive role) or in the absence of reactive measures. This differentiation is pragmatic since reactive measures can be based on concrete existing problems and can thus be more fine grained, whereas proactive measures support a goal in general.

B. AGQM Inputs

Reports are generated by static analysis tools such as PMD and imported into CoSEEEK. This generation can be triggered manually by developers or as part of an automatic build process. Via the sensors and the event architecture, these external tools are integrated into CoSEEEK so that the reports are available to all modules. The reports also contain metric violations that are processed by CoSEEEK's Rules Processing module, producing unified reports abstracted from the concrete tools and containing proposed concrete measures attributed to metric violations. The measures are not ordered or sorted according to importance or urgency and thus not yet arranged for automatic selection. These reports furnish the input for the AGQM module.

C. AGQM Process

At the beginning of a project, phase, or iteration, a quality manager assigns a point weighting to each goal (implying its importance) and chooses a bidding strategy for the agent managing that goal. The points are used by agents for negotiating proposed measures. The AGQM process invokes a proactive as well as reactive selection mechanism that results in a measure proposal.

Quality goals can be conflicting, and determining the appropriate balance is project-specific. Thus, a competitive bidding process among agents was chosen for proactive measures, whereas a cooperative voting process was chosen for reactive measures. The competitive bidding allows agents with greater importance to definitively have opportunities to support their goal with measures, in contrast to voting where agent majorities might win. That way a group of lower

priority goal agents does not hinder a higher priority goal from asserting influence. The strategies enable agents to win opportunities earlier or later in an iteration cycle.

The reactive voting process is cooperative since a potentially large number of concrete reactive measures based on metric violations are possible for a limited number of quality opportunity slots (Q-slots), and those measures that will have the greatest overall quality impact are favored. The agents cooperatively vote on the measures from the reports received. Via the structure shown in Figure 2, each agent determines for each measure if a measure belongs to a metric that is related to the agent's goal. An agent's points are then distributed (currently uniformly) across all measures associated to its goal.

The proactive section utilizes the violations in the reports for the calculation of the different KPIs, QKPIs, and GKPIs. If there are deviations at the goal level, measures attributed to that goal are 'activated', meaning they are used by the agents in the proactive section. Each agent bids for a unique set of proactive measures and the highest bid wins, elevating that measure to a proposal. In this process, not just the number of distributed points differentiates between the goals, but also the strategy chosen by the agents. The strategies influence how an agent increases or decreases its bids after winning or losing for the next bidding process. Choosing a defensive strategy for an agent will increase the likelihood that a proposal of its associated measures will occur in later phases of the iteration. This behavior occurs because in early sessions the agents with more aggressive strategies will place much higher bids. The defensive agent can then place the winning bid later when the aggressive agents run out of points.

The event trigger for execution of the AQGM process is defined by the availability of a Q-slot. At this point, a software engineer has an opportunity for the execution of a quality measure. Note that the detection of these Q-slots, human interactions, and the integration of the activities in the software engineer's concrete workflow are out-of-scope for this paper. When a Q-slot occurs, it has to be determined if a proactive or a reactive measure should be proposed. This can be configured by defining a proactive-to-reactive ratio. If no metrics are yet present and reactive measures are unavailable, then no question or goal deviation is detectable since there is no basis for their calculation. In this case, one of the pre-defined proactive measures is selected. Section IV will evaluate a concrete scenario utilizing this approach.

III. AGQM

This section describes the technical realization of the different components and details the internal structure of the AGQM module.

The communication of the different modules via events is implemented utilizing an XML implementation of the tuple space paradigm [8] with the eXist XML database [15] for event storage. Communication with the space was realized via web services using Apache CXF. For event extraction from various tools such as PMD, the Hackystat framework [9] was chosen. Esper was used for complex event processing. The Rules Processing module was implemented using JBoss Drools. The AGQM module, which is the focus of this paper, was implemented via the FIPA-compliant [18] Jade Framework [2]. The configuration for GQM and agents as well as measure/violation lists are XML-based.

The agent structure is defined as depicted in Figure 3. The AGQM agent is responsible for the management of the agent module. It instantiates the other agents and determines if a reactive or a proactive measure will be proposed. For each defined goal, one goal agent is instantiated. In the proactive section, the goal agents communicate with the session agent to realize the bidding process. The session agent takes the role of the "buyer" and thus selects the proactive measure from the goal agent with the highest bid. Each goal agent places bids according to its strategy. For the initial implementation, basic strategies were used. The three strategies 'offensive', 'balanced', and 'defensive' influence the starting bid of the agents as well as win-or-lose adaptation based on the last session. The strategy pattern allows these algorithms to vary. If insufficient points are left for the intended bid, the agent bids the rest of its points. If an agent has no points left, it cannot place bids anymore until all agents have no points left, whereupon all points are reset to their initial value. Each agent has a list of proactive measures it could offer, which is pre-filtered to contain only measures whose attributed GKPIs are violated. Thus, goals that are known to be at risk are elevated to participation status in the bidding. If no report containing GKPI violations was received, the list contains all proactive measures attributed to a goal.

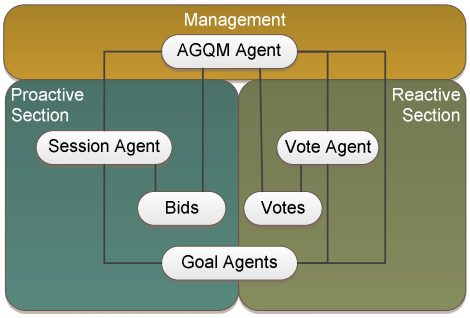

Figure 3. Agent Structure

The reactive section is realized via the vote agent. Each time a report is received, the vote agent creates a weighted list of reactive measures using the report. To elicit the weight of each measure, the vote agent communicates with the goal agents. For each measure, a goal agent evaluates if that measure is attributed to its goal via the aforementioned connection of measures, metrics, KPIs, and goals. In each voting process, a goal agent distributes all of its assigned points (initially allocated at the beginning of the iteration) uniformly across all measures in the current report that are attributed to its goal. If multiple agents vote on one measure, the points are aggregated. If no report was yet received, the voting process cannot be conducted. In that case, a proactive measure is substituted.

IV. EVALUATION

Two types of evaluations of the approach were conducted. A concrete scenario was created in a lab environment to analyze the suitability of the results. To determine any technical limitations, performance and scalability were measured. The test configuration consisted of one computer with an Intel Core i7 Q820 1.73 GHz processor and 6GB RAM. The software used was Windows 7 64-bit, the Java Runtime Environment 1.5.0_20, Apache CXF 2.2.4., eXist 1.2.6 (rev. 9165) and Jade 3.7. The tests were executed in a virtual machine (VMware Player 3.0.1 build-227600) assigned two processor cores and 4GB RAM.

A. Scenario

For clarity, all scenario parameters have been selected as follows. Four goals functionality, reliability, maintainability, and performance were defined, each with a set of questions, metrics, and measures. Table I shows the selected items for reliability (due to the size of the XML, only the relevant data is shown). The goal has two questions dealing with complexity and defect ratio. Complexity is then further split into the two KPIs: code complexity, having two associated metrics; and design complexity, having only one metric. For simplicity, defect ratio has only one question with only one KPI and metric. Furthermore, one measure is assigned to each metric. The other goals were handled correspondingly.

TABLE I. EXTENDED GQM STRUCTURE FOR RELIABILITY (EXTRACTED FROM XML)

Goal: G:RELIABILITY
GKPI: GKPI:RELIABILITY
Questions: Q:CPLX;Q:DEF-RATIO
Proactive Measures:M:P:Cond code reviews; M:P:Expand test coverage
QKPIs:QKPI:Complexity; QKPI:DefectRatio
KPIs:KPI:Code Complexity; KPI:Design Complexity; KPI:DefectRatio
Metrics: MET:Cyclomatic Complexity; MET:Complexity; MET:CouplingFactor; MET:DefectRatio; MET:ClassFanOut
Measures:M:R:Refactor code; M:R:Refactor design; M:R:CodeReview

The scenario was designed to show that the agents select adequate measures for an iteration in a development project, in which the main emphasis is developing new functionality (i.e. that features exist) followed by reliability and maintainability that are weighted equally. The least important of the four main goals is performance. Note that the actual semantic meaning of the goals is a human matter - see Figure 2 for the system construct. Accordingly, the agent points and strategies were selected as depicted in TABLE II.

TABLE II. AGENT CONFIGURATION

Agent	Points	Strategy
Functionality	100	Offensive
Reliability	80	Balanced
Maintainability	80	Balanced
Performance	60	Defensive

The iteration considered in the scenario is assumed to allow 20 Q-slots. The selection ratio between proactive and reactive measures was configured so that 70% of the proposed measures would be reactive. Initially no report containing metric violations is available. It is anticipated that after the third Q-slot such a report is received. Thus, the first three slots necessarily become substituted with proactive measures. Table III shows the proposed quality measures generated for a run across all 20 slots. Proactive measures are identified by the prefix "M:P:" and reactive measures by "M:R:".

TABLE III. PROPOSED QUALITY MEASURES

Slot	Quality Measure
1	M:P:Analyze reuse possibilities
2	M:P:Update architecture/design documents
3	M:P:Expand test coverage
4	M:P:Analyze modularity
5	M:P:Analyze reuse possibilities
6	M:R:Refactor code
7	M:R:Refactor design
8	M:R:Address use case coverage
9	M:R:Implement error handling functionality
10	M:R:Address acceptance testing coverage
11	M:R:Perform code review
12	M:R:Optimize throughput
13	M:P:Expand test coverage
14	M:R:Opotimize throughput
15	M:P:Expand test coverage
16	M:R:Optimize memory usage
17	M:R:Tune code/design
18	M:R:Refactor for performance
19	M:R:Increase comment ratio
20	M:P:Analyze bottlenecks

To determine the impact of the strategies in conjunction with the distribution of points in the proactive section, Table IV shows the agents bids for the slots, in which proactive measures were proposed. The numbers in parenthesis indicate the bid an agent would have placed according to its strategy when insufficient points were available.

TABLE IV. AGENTS BIDS

Slot	Winner	FUNC	REL	MAINT	PERF
1	FUNC	**35**	24	24	15
2	FUNC	**31**	28	28	17
3	REL	28	**32**	32	19
4	MAINT	34	28	**37**	21
5	FUNC	**34(41)**	32	32	23
13	MAINT	0	37	**37**	25
15	REL	0	**43**	32	28
20	PERF	0	25(37)	26(37)	**31**

The results in TABLE III. correlate with the expected temporal arrangement of the proposed measures, where

functionality measures should be favored and more probable towards the beginning of the iteration and performance measures toward the end of the iteration. Reliability and maintainability should be more probable somewhere in-between. The scenario is not detailed and broad enough to prove the applicability for the majority of SE real world use cases, yet it shows the potential of the approach towards automated GQM and SQM. Future work will include industrial trials of this approach with empirical results.

B. Performance Measurements

All performance measurements were conducted five times consecutively, taking the average of the last three measurements.

The latency for vote list creation varying the numbers of measures and goals is depicted in Table V. The results show that the number of measures has a greater impact on the latency versus increases in the number of goal agents, when measurement inaccuracies regarding the smaller values are disregarded.

TABLE V. AVERAGE VOTE LIST CREATION LATENCY (MS) VS. GOALS AND MEASURES

Measures	50	100	500	1000
5 Goals	111	194	273	924
10 Goals	113	160	815	1927
15 Goals	110	263	787	2090
50 Goals	92	317	842	2453
100 Goals	91	342	864	3003

A second measurement considered the measure proposal latency for a slot. It was assumed that for every goal exactly one proactive measure was defined, thus only the number of goals was of interest. All agents were given an offensive strategy and 100 points. For reactive measures the vote list was already created and sorted, from which only the first position is taken. The results are shown in Table VI.

TABLE VI. AVERAGE MEASURE PROPOSAL LATENCY (MS) VS. GOALS

	5 Goals	10 Goals	15 Goals	50 Goals	100 Goals
Proactive	47	51	45	65	3211
Reactive	40	325	338	492	665

The reactive part shows the overhead of increasing agents for retrieving the top measure from the vote list. The proactive part remains constant for low goal numbers and then reaches an inflection point with a large number of goal agents. One possible explanation is extended bidding and thrashing with thread-based agents - this should be further investigated.

In summary, the performance of the current implementation appears sufficient for use in SEEs when the number of goals and measures used are within expected limitations. Performance could become an issue in large teams or projects, or when large numbers of reactive measures are triggered. One way to address this would be to tune the Rules Processing Module to limit the number of reactive measures for which voting takes place. As to goal scalability, a large number of goals and goal agents would also imply a high degree of configuration overhead for a quality manager, thus likely naturally limiting the number of goals. Should nevertheless a large number of goals be desirable, distributing the agents could be considered.

V. RELATED WORK

The combination of GQM with agents has been used for providing automated support for GQM plan creation [5][10][7] and for the computation of values for questions and goals [21][22]. In [7], a goal-driven use case method is utilized to elicit requirements. A set of agents assists the user in identifying goals and questions that are then used by another agent to obtain metrics. The collection of the measurement data and the creation of the measurement plan are then executed by two other agents. The ISMS (Intelligent Software Measurement System) [5][10] follows a similar approach using different groups of agents for user assistance and determination of different parts of the GQM plan. In [21][22], agents are used in the requirements process of the SW-CMM (Software Capability Maturity Model) model. The focus is the measurement and analysis of software processes using agents and fuzzy logic.

The approach in [11] aims for automated user assistance in GQM plan creation and execution but does not utilize agent technology. A tool was developed which allows the creation of GQM plans using predefined forms as well as the verification of the structural consistency of the plan and the reuse of its components. Furthermore, the tool supports data interpretation and analysis through aggregation of collected data. [12] is an extension of that approach, integrating GQM more tightly with a development process to support GQM plan creation by an explicit process model.

For better integration of the GQM technique into the project flow via automation, different approaches were considered. [14] aims at integrating measurement programs as well as data collection into explicit process models while [16] provides an object oriented-process model whose target is measurement. [3] proposes the usage of process models for the creation of GQM plans. The tool Prometheus [24] links executive plans with process models.

An approach that extends the GQM technique is presented in [6], which adds concepts such as entities, attributes, and units. cGQM [13] proposes the use of the Hackystat framework for GQM, applying continuous measurement with short feedback loops.

Other applications of agent technology include its utilization for automatic information retrieval [20], process monitoring [25], or collaboration support [23].

In contrast to the aforementioned approaches, CoSEEEK's AGQM process applies a combination of these techniques to live SEEs along with active SQM countermeasure proposals to developers. Agent technology is used differently in that the aim is neither user assistance in GQM plan creation nor assistance in interpretation of measurement results. It is rather the fully automatic monitoring of goal fulfillment and the automatic assignment of quality measures for different types of quality deviations.

VI. CONCLUSION AND FUTURE WORK

The many challenges due to SE project uniqueness, dynamics, and limited quality budgets and opportunities necessitate greater automation and tighter integration of quality measurement techniques in SEEs. CoSEEEK's AQGM process addresses this by automating and extending GQM for on-the-fly use in live SEEs, leading to more timely, suitable, and thus effective proactive and reactive quality measures.

Currently proactive and reactive quality countermeasures are usually not based on timely automated measurement data, nor are their relations to quality goals necessarily consciously apparent. The AGQM process systematically harmonizes quality attributes, goals, metrics, and countermeasures and supports the automation of aspects of SQM. By assigning goal-monitoring responsibility to agents, goals are not neglected and relevant measures are proposed in accordance with goal importance. The GQM technique provides the systematic basis while the extension with assigned measures enable these to be selected and assigned at temporally relevant points in SEEs based on causal data.

Future work will assess the effectiveness of the approach via case studies in industrial settings. Work is required to address the appropriate planning, determination, and placement and frequency of Q-slots in industrial settings. Incorporating effort estimation and available time and resource aspects, as well as assignment matching to user profiles (background, experience) will be considered. More complex agent strategies in addition to systematic detection of human expertise situations will also be researched.

ACKNOWLEDGMENTS

The authors wish to acknowledge Muhammer Tüfekci and Stefan Lorenz for their assistance with the implementation and evaluation. This work was sponsored by BMBF (Federal Ministry of Education and Research) of the Federal Republic of Germany under Contract No. 17N4809.

REFERENCES

[1] Basili, V., Caldiera, G., and Rombach, H.D., "Goal Question Metric Approach," Encycl. of Software Engineering, John Wiley & Sons, Inc., 1994, pp. 528-532.

[2] Bellifemine, F., Poggi, A., and Rimassa, G., "JADE - A FIPA-compliant Agent Framework," Proc. 4th Intl. Conf. and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents. London, 1999.

[3] Broeckers A., Differding C., and Threin G., "The Role of Software Process Modeling in Planning Industrial Measurement Programs," In Proc. of Int. Metrics Symposium, Berlin 1996.

[4] Brooks, F. P. Jr., "The Mythical Man-Month," Addison-Wesley Professional, ISBN: 0201835959, 1995.

[5] Chen, T., Homayoun Far, B., and Wang, Y., "Development of an Intelligent Agent-Based GQM Software Measurement System," doi:10.1.1.146.5373.

[6] De Panfilis, S., Kitchenham B. and Morfuni N., "Experiences introducing a measurement program," Information and Software Technology 39(11) (1997) pp. 745-754.

[7] FanJiang, Y.-Y and Wu, C.-H. "Towards a Multi-agents Architecture for GQM Measurement System," 9th Int'l Conf. on Hybrid Intelligent Systems, 2009, pp. 277-280.

[8] Gelernter, D., "Generative communication in Linda," ACM Transactions on Programming Languages and Systems, vol. 7, nr. 1, Jan. 1985, pp. 80-112.

[9] Johnson, P. M., "Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System," Proc. of the First International Symposium on Empirical Software Engineering and Measurement, IEEE, 2007, pp. 81-90.

[10] Junling Huang Far, B.H., "Intelligent software measurement system (ISMS)," Canadian Conf. on Electrical and Computer Engineering, 2005, pp. 1033 – 1036.

[11] Lavazza, L., "Providing automated support for the GQM measurement process," Software, IEEE, Volume 17, Issue 3, May-June 2000, pp.:56 – 62, doi: 10.1109/52.896250.

[12] Lavazza, L. and Barresi, G., "Automated support for process-aware definition and execution of measurement plans," Proc. of the 27th Int'l Conf. on Software engineering, 2005, pp. 234 – 243.

[13] Lofi C., "cGQM - Ein zielorientierter Ansatz für kontinuierliche, automatisierte Messzyklen," Proc. of the 4th National Conf. on Software Measurement and Metrics (DASMA MetriKon 2005), 2005.

[14] Lott C. M. and Rombach H. D.. "Measurement-based guidance of software projects using explicit project plans,", Information and Software Technology, June/July 1993.

[15] Meier, W., "eXist: An Open Source Native XML Database," Web, Web-Services, and Database Systems, Springer Berlin / Heidelberg, 2009 pp. 169-183.

[16] Morisio M., "Measurement Processes are Software Too," Journal of Systems and Software, December 1999.

[17] Oberhauser, R.. "Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments," In "Semantic Web", Gang Wu (ed.), In-Tech, Vienna, Austria, 2010.

[18] O'Brien, P. D. and Nicol, R.C., "FIPA — Towards a Standard for Software Agents," BT Technology Journal 16(3), 1998, pp. 51-59.

[19] Ozkaya, I., Bass, L., Sangwan, R. S., and Nord, R. L. 2008. "Making Practical Use of Quality Attribute Information," IEEE Softw. 25, 2 (Mar. 2008), pp. 25-33. DOI= http://dx.doi.org/10.1109/MS.2008.39.

[20] Pelletier, S.-J., Pierre, S., and Hoang, H.H., "Modeling a Multi-Agent System for Retrieving Information from Distributed Sources," in Journal of Computing and Information Technology, Vol 11, No 1, 2003, pp. 15-39.

[21] Seyyedi, M.A., Teshnehlab, M., and Shams, F., "Measuring software processes performance based on the fuzzy multi agent measurements," In Proc. Intl Conf. on Information Technology: Coding and Computing (ITCC'05) - Volume II, 2005. IEEE Computer Society, Washington, DC, 410-415.

[22] Seyyedi, M.A., Shams, F., and Teshnehlab, M., "A New Method For Measuring Software Processes Within Software Capability Maturity Model Based On the Fuzzy Multi- Agent Measurements," Proc. of World Academy Of Science, Engineering and Technology Vol. 4, 2005, pp. 257-262.

[23] Tan, W., Chen, R., Shen, W., Zhao, J., and Hao, Q., "An Agent-Based Collaborative Enterprise Modeling Environment Supporting Enterprise Process Evolution," Computer Supported Cooperative Work in Design III, Springer, pp. 217-226.

[24] Visaggio, G., "Process Improvement Through Data Reuse," IEEE Software 11(4), July 1994, pp. 76-85.

[25] Wang, M., Wang, H., and Xu, D., "The design of intelligent workflow monitoring with agent technology," in Knowledge-Based Systems, Vol. 18, Issue 6, 2005, pp. 257-266.